Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии» (ИУ7)

Отчет по лабораторной работе №6 по курсу «Анализ алгоритмов»

«Методы решения задачи коммивояжера»

Группа: ИУ7-53Б

Студент: _____ Дьяченко А. А.
(Подпись, дата) (Фамилия И. О.)

Преподаватель: _____ Строганов Д. В.
(Подпись, дата) (Фамилия И. О.)

Москва, 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В 1831 г. в Германии вышла книга под названием «Кто такой коммивояжер и что он должен делать для процветания своего предприятия». Одна из рекомендаций этой книги гласила: «Важно посетить как можно больше мест возможного сбыта, не посещая ни одно из них дважды». По-видимому, это была первая формулировка задачи коммивояжера. Наиболее употребляемая формулировка ЗК состоит в следующем. Заданы список городов некоторого региона и таблица попарных расстояний между ними. Требуется найти замкнутый (т. е. начинающийся и заканчивающийся в одном и том же городе) маршрут коммивояжера, проходящий через все города, причем входящий в каждый город и выходящий из каждого города по одному разу и имеющий минимальную длину [**РёРҳРњРѳРёРҳРұ1989РчРѳРұРѳСЛЬРѳ**].

Такая задача может быть решены при помощи полного перебора вариантов и эвристических алгоритмов. Алгоритмы, основанные на использовании эвристического метода, не всегда приводят к оптимальным решениям. Однако для их применения на практике достаточно, чтобы ошибка прогнозирования не превышала допустимого значения.

Целью данной лабораторной работы является сравнительный анализ метода полного перебора и метода на базе муравьиного алгоритма в соответствии с вариантом.

Вариант: ор-граф, с элитными муравьями, карта перемещения для воздухоплавателей (время с учётом направления — по ветру или против ветра; горы нужно огибать; в привязке к карте). Гамильтонов цикл.

Задачи лабораторной работы:

— описать схемой алгоритма и реализовать метод полного перебора для решения задачи коммивояжера;
— описать схемой алгоритма и реализовать метод решения задачи коммивояжёра на основе муравьиного алгоритма;
— указать преимущества и недостатки реализованных методов;
— провести сравнительный анализ двух рассмотренных методов решения задачи коммивояжера;
— подготовить отчет по лабораторной работе.

1 Аналитическая часть

1.1 Алгоритм полного перебора

Для решения задачи коммивояжера алгоритм полного перебора предполагает рассмотрение всех возможных путей в графе и выбор наименьшего из них. Смысл перебора состоит в том, что перебираются все варианты объезда городов и выбирается оптимальный, что гарантирует точное решение задачи. Однако, при таком подходе количество возможных маршрутов факториально возрастает — сложность алгоритма равна $n!$.

1.2 Муравьиный алгоритм

Муравьиный алгоритм — метод решения задач коммивояжера на основании моделирования поведения колонии муравьев.

Каждый муравей определяет для себя маршрут, который необходимо пройти на основе феромона, который он получает во время прохождения. Каждый муравей оставляет феромон на своем пути, чтобы остальные муравьи по нему ориентировались. В результате при прохождении каждым муравьем различного маршрута наибольшее число феромона остается на оптимальном пути. Распределение феромона по пространству передвижения муравьев является своего рода динамически изменяемой глобальной памятью муравейника. Любой муравей в фиксированный момент времени может воспринимать и изменять лишь одну локальную ячейку этой глобальной памяти [**СЇСЂР«РўРçРѳ2003РёСЋСЃРѳРўСÆРчР„СЌРх**].

2 Конструкторская часть

В этом разделе будут представлены схемы реализуемых алгоритмов.

2.1 Полный перебор

На рисунке 1 приведена схема алгоритма решения задачи коммивояжера полным перебором.

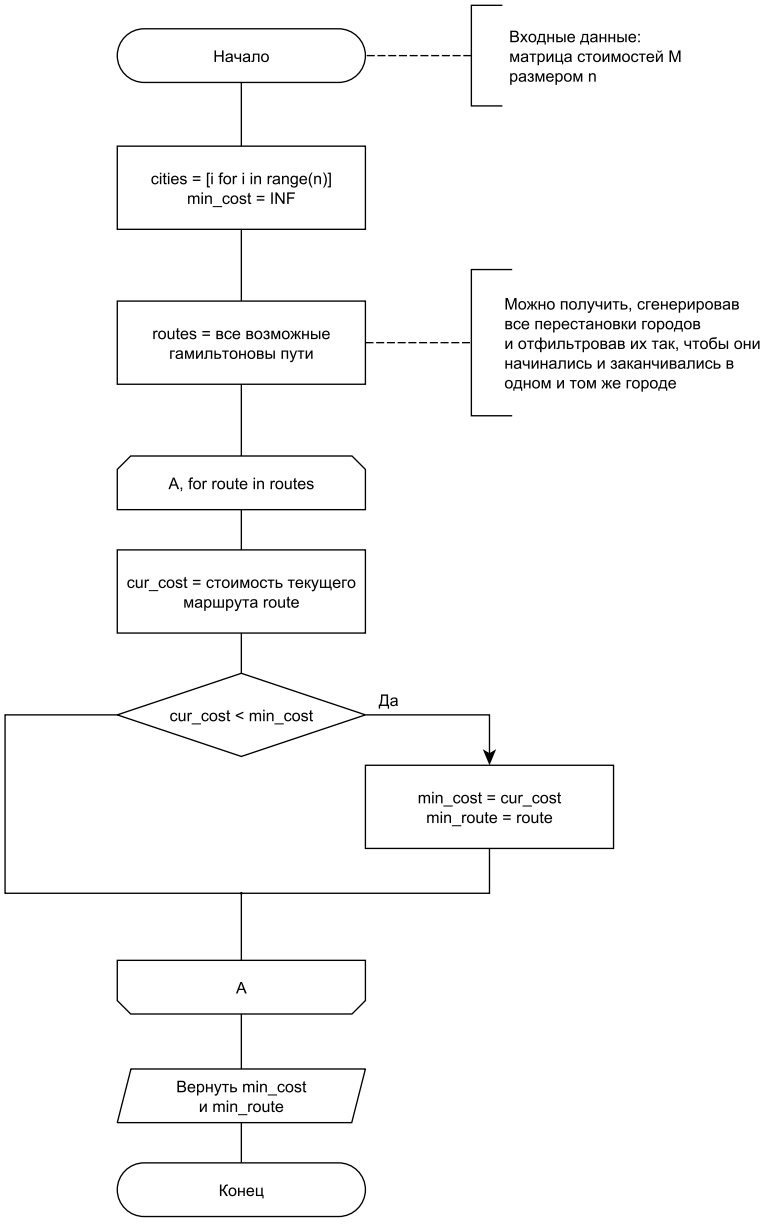

Рисунок 1 – Полный перебор

Преимущество алгоритма: оптимальный путь будет найден всегда.
Недостаток: скорость.

2.2 Муравьиный алогритм

На рисунке 2 приведена схема алгоритма решения задачи коммивояжера муравьиным алгоритмом.

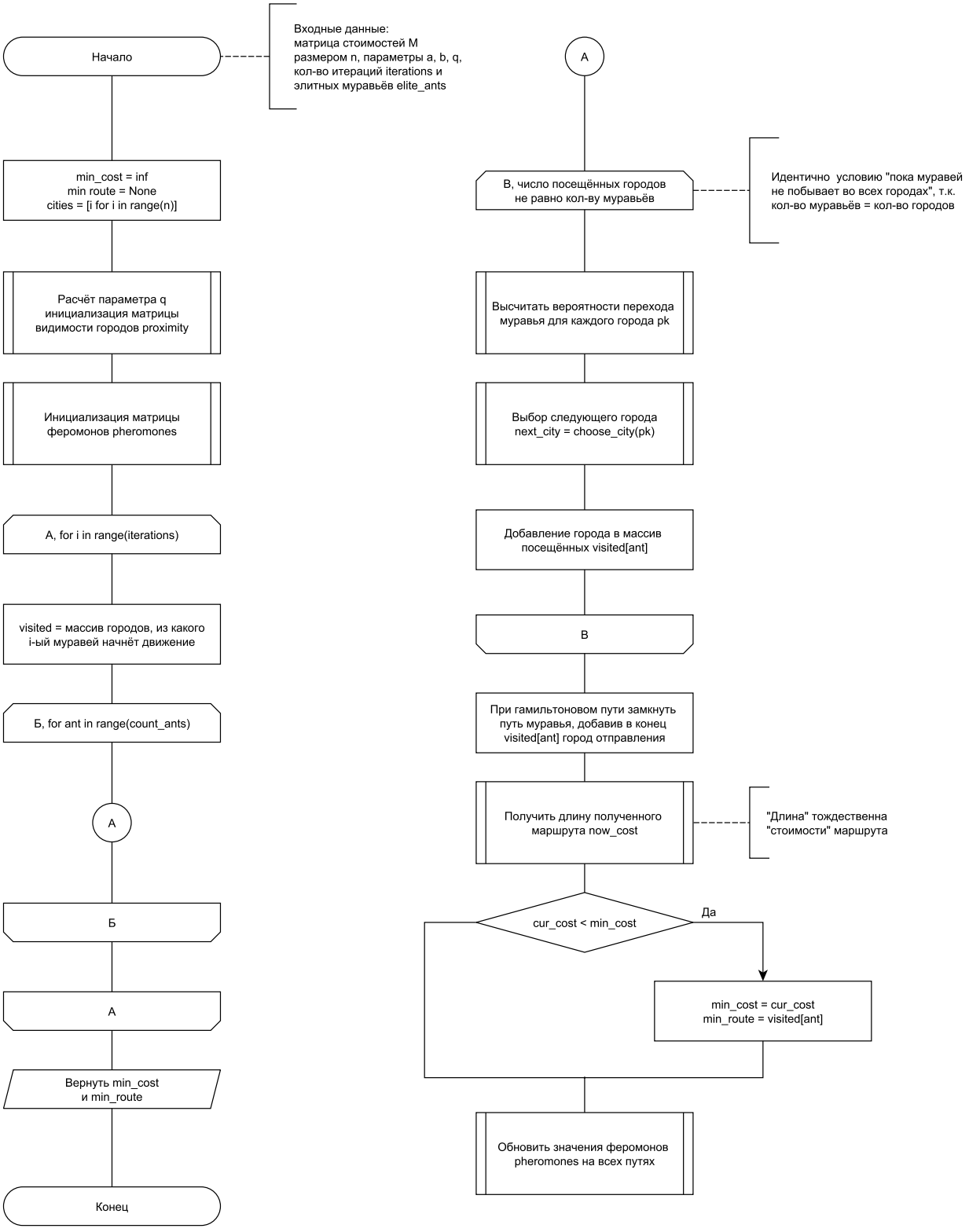

Рисунок 2 – Муравьиный алгоритм

На рисунке 3 приведена схема алгоритма вычисления вероятностных переходов.

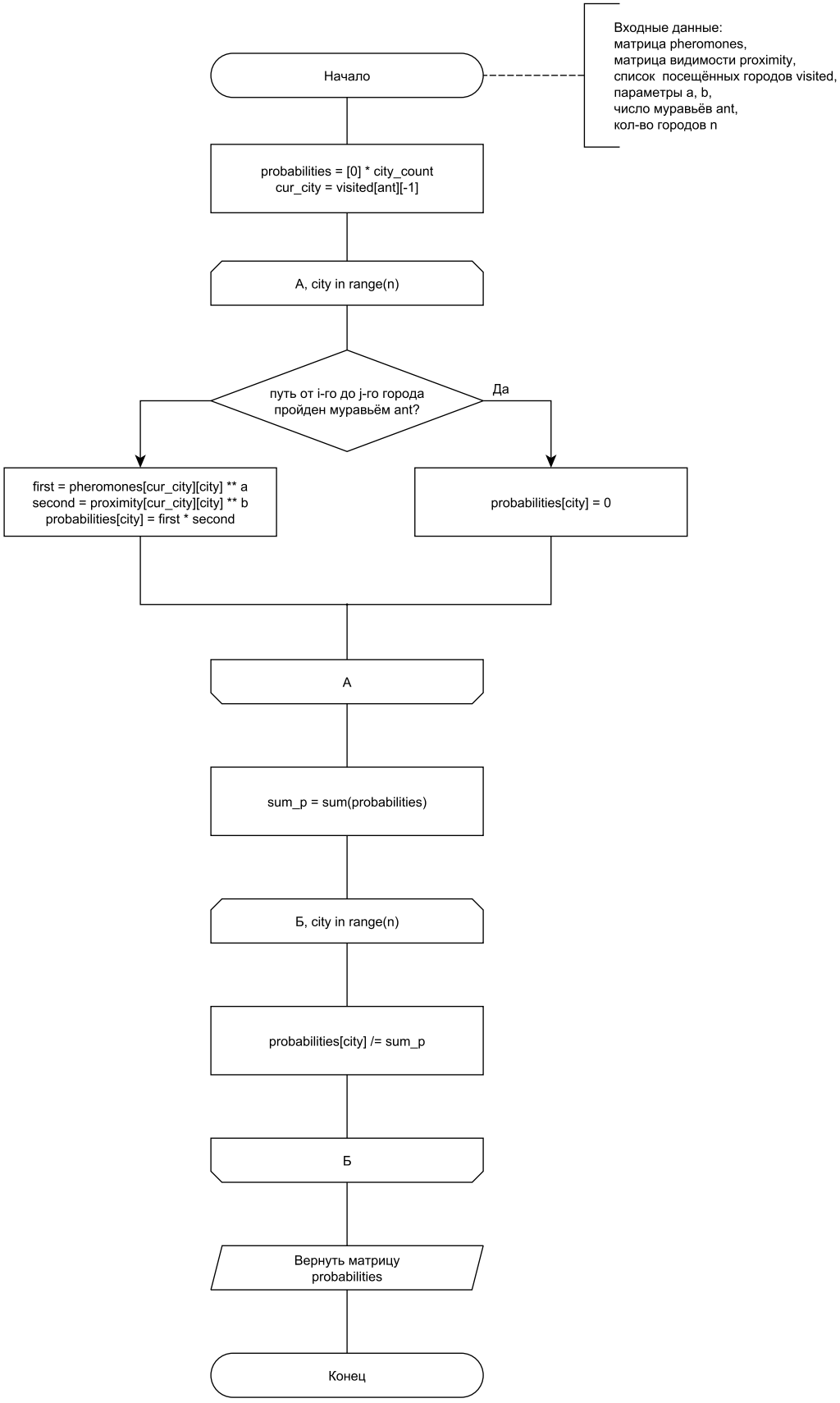

Рисунок 3 – Вычисление вероятностных переходов

На рисунке 4 приведена схема алгоритма обновления кол-ва феромонов на путях.

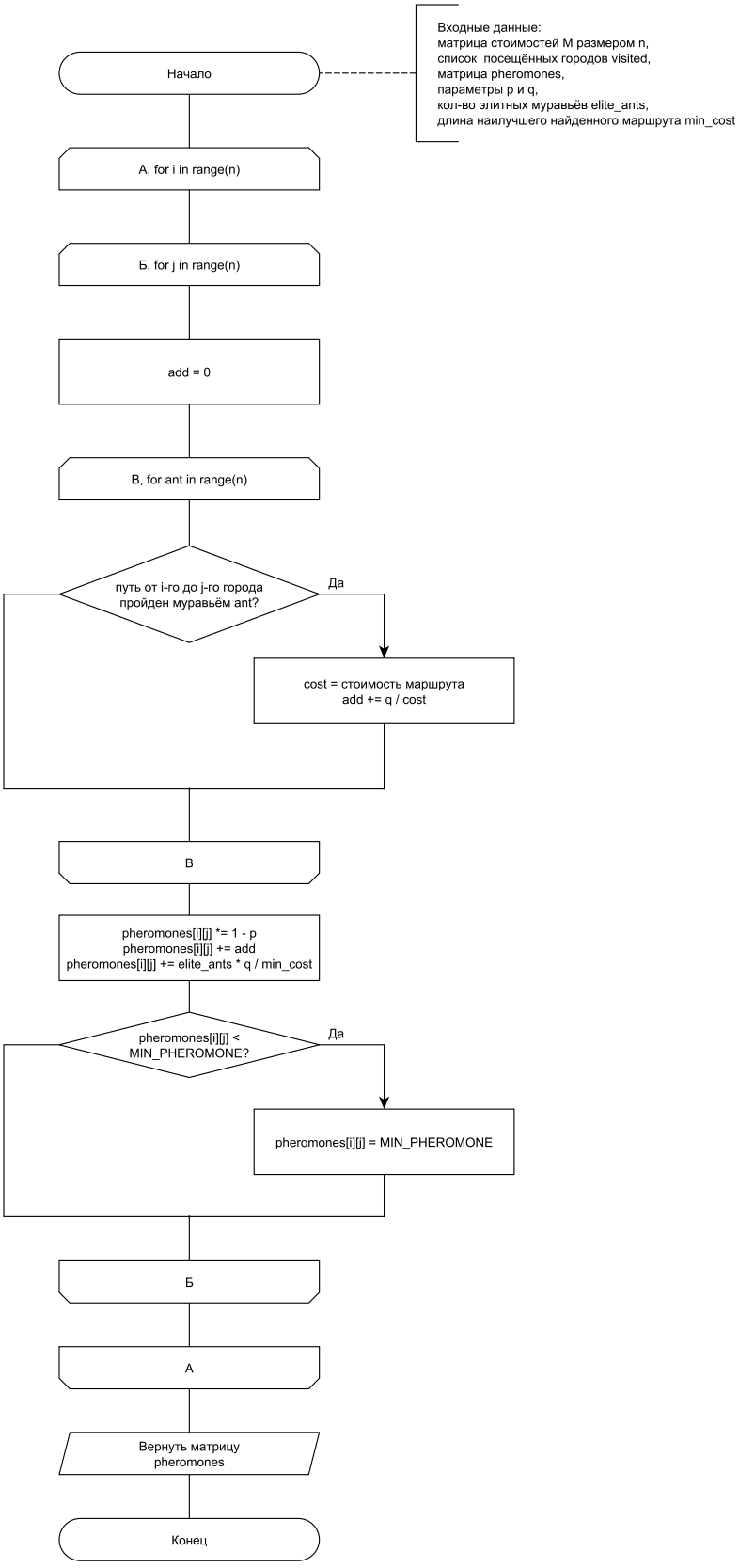

Рисунок 4 – Обновление феромона

Обратите внимание: по варианту необходимо рассмотреть муравьиный алгоритм с элитными особями. Элитный муравей усиливает ребра наилучшего маршрута, найденного с начала работы алгоритма [**СЇСЪР«РўРçРѳ2003РёСЋСЃРѳРўСÆРчР„СЌРх**]. Именно поэтому к значению $pheromones[i][j]$ добавляется значение $\frac{elite\_ants \cdot q}{min\_cost}$. В алгоритме без элитных муравьёв этого слагаемого нет.

Остальные схемы подпрограмм, указанные на рисунке 2 не приведены, т.к. они зависят от конкретной реализации. Подпрограммы рисунках 3 и 4, в свою очередь, от реализации к реализации не меняются (за исключением модификаций метода, таких как добавление элитных муравьёв).

Преимущество метода: скорость.

Недостаток: для нахождения оптимального пути требуется увеличение итераций и подбор параметров опытным путём.

3 Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к ПО

На вход подаётся матрица стоимостей. На выходе требуется получить минимальный путь — последовательность вершин, которые необходимо посетить. Т.к. по варианту необходимо найти Гамильтонов путь, первая и последняя вершина в последовательности всегда должна совпадать.

3.2 Средства реализации

В качестве языка программирования для реализации лабораторной работы был выбран Python. Данный выбор обусловлен наличием библиотеки *time* [**time**] для замера времени выполнения алгоритмов.

3.3 Сведения о модулях программы

Программа состоит из следующих программных модулей:

1) matrix.py — модуль с функциями обработки матриц;
2) ant.py — модуль с реализацией муравьиного алгоритма;
3) bruteforce.py — модуль с реализацией метода полного перебора;
4) main.py — главный модуль с меню и функцией параметризации;
5) time.py — модуль замера времени алгоритмов;
6) plot.py — модуль создания графического представления значений.

3.4 Реализация алгоритмов

В листинге 1 приведена реализация метода полного перебора.

Листинг 1 – Полный перебор

```
import itertools


def calculate_cost(matrix, route):
    return sum(matrix[start_city][end_city] for start_city,
       end_city in zip(route, route[1:]))


def get_all_routes(n):
    cities = list(range(n))
    permutations = list(itertools.permutations(cities))
    valid_paths = [p + (p[0],) for p in permutations]
    return valid_paths


def brute_force(n, matrix):
    routes = get_all_routes(n)
    min_cost = float('inf')
    min_route = None
    for route in routes:
        current_cost = calculate_cost(matrix, route)
        if current_cost < min_cost:
            min_cost = current_cost
            min_route = route
    return min_route, min_cost


test_matrix = [[0, 10, 15, 20, 25], [10, 0, 35, 5, 10], [15, 35, 0,
    30, 5], [20, 5, 30, 0, 15], [25, 10, 5, 15, 0]]

n = 5

min_route, min_cost = brute_force(n, test_matrix)
```

В листинге 2 приведена реализация муравьиного алгоритма.

Листинг 2 – Муравьиный алгоритм

```
def get_cost(matrix, route):
    cur_cost = 0
    for num in range(len(route) - 1):
        start_city = route[num]
        end_city = route[num + 1]
        cur_cost += matrix[start_city][end_city]
    return cur_cost


def get_all_routes(cities):
    routes = []
    for route in itertools.permutations(cities, len(cities)):
        route = list(route)
        route.append(route[0])
        routes.append(route)
    return routes


def get_q(matrix, size):
    q = 0
    count = 0
    for i in range(size):
        for j in range(i + 1, size):
            q += matrix[i][j] * 2
            count += 2
    return q / count


def get_proximity(matrix, size):
    proximity = [[0 for i in range(size)] for j in range(size)]
    for i in range(size):
        for j in range(i):
            proximity[i][j] = 1 / matrix[i][j]
            proximity[j][i] = proximity[i][j]
    return proximity
```

Листинг 3 – продолжение листинга 2

```
def get_visited_cities(cities, count_ants):
    visited = [[] for _ in range(count_ants)]
    for ant in range(count_ants):
        visited[ant].append(cities[ant])
    return visited


def get_probability(pheromones, proximity, visited, a, b, ant,
   city_count):
    probabilities = [0] * city_count
    now_city = visited[ant][-1]
    for city in range(city_count):
        if city not in visited[ant]:
            p = (pheromones[now_city][city] ** a) * (proximity[
               now_city][city] ** b)
            probabilities[city] = p
        else:
            probabilities[city] = 0
    sum_p = sum(probabilities)
    for city in range(city_count):
        probabilities[city] /= sum_p
    return probabilities


def choose_city(probabilities):
    num = random()
    value = 0
    for i in range(len(probabilities)):
        value += probabilities[i]
        if value > num:
            return i


def check_route(i, j, route):
    if i in route:
        index_i = route.index(i)
        if j == route[index_i + 1]:
            return True
    return False
```

Листинг 4 – продолжение листинга 3

```
1
2
3  def update_pheromone(visited, matrix, pheromones, city_count, p, q,
       elite_ants_count, min_cost):
4      for i in range(city_count):
5          for j in range(city_count):
6              add = 0
7              for ant in range(city_count):
8                  if check_route(i, j, visited[ant]):
9                      cost = get_cost(matrix, visited[ant])
10                     add += q / cost
11             pheromones[i][j] *= (1 - p)
12             pheromones[i][j] += add + elite_ants_count * q /
                   min_cost
13             if pheromones[i][j] < MIN_PHEROMONE:
14                 pheromones[i][j] = MIN_PHEROMONE
15     return pheromones
```

Листинг 5 – продолжение листинга 4

```
def ant_algorithm(matrix, size, a, b, p, iterations,
    elite_ants_count):
    cities = [i for i in range(size)]
    min_cost = inf
    min_route = None
    q = get_q(matrix, size)
    proximity = get_proximity(matrix, size)
    pheromones = [[START_PHEROMON for i in range(size)] for j in
        range(size)]
    count_ants = size
    for _ in range(iterations):
        visited = get_visited_cities(cities, count_ants)
        for ant in range(count_ants):
            while len(visited[ant]) != count_ants:
                pk = get_probability(pheromones, proximity, visited
                    , a, b, ant, size)
                next_city = choose_city(pk)
                visited[ant].append(next_city)
            visited[ant].append(visited[ant][0])
            cur_cost = get_cost(matrix, visited[ant])
            if cur_cost < min_cost:
                min_cost = cur_cost
                min_route = visited[ant]
        pheromones = update_pheromone(visited, matrix, pheromones,
            size, p, q, elite_ants_count, min_cost)
    return min_route, min_cost
```

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялся замерный эксперимент:

— операционная система Windows 11;

— память 16 ГБ;

— процессор 3,6 ГГц 6-ядерный процессор AMD Ryzen 5000 series 5.

Замеры проводились на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только интегрированной средой разработки и непосредственно выполняемой программой.

4.2 Пример работы программы

На рисунке 5 представлен пример работы программы.

Рисунок 5 – Пример работы программы

4.3 Время выполнения реализованных алгоритмов

Функция perf_counter из библиотеки time [**time**] возвращает время в секундах — значение типа float. Является самым точным методом замера коротких промежутков времени [**perf**].

Замеры проводились 100 раз для матриц стоимостей, заполненных случайным образом, размером от 2 до 10.

Результаты измерения времени приведены на рисунке 6.

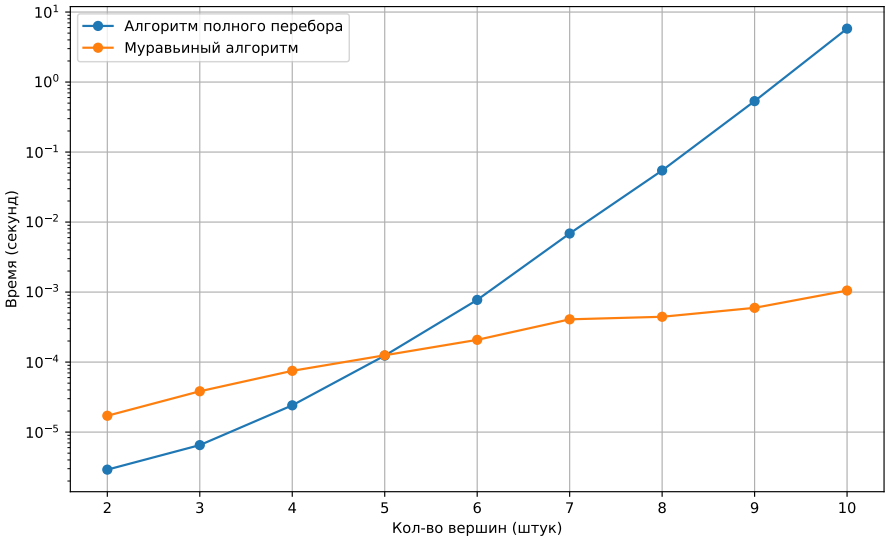

Рисунок 6 – Время выполнения алгоритмов

4.4 Параметризация

Целью проведения параметризации является определение таких комбинаций параметров, при которых муравьиный алгоритм дает наилучшие результаты.

В результате автоматической параметризации будет получена таблица со следующими столбцами:

— коэффициент видимости α — изменяющийся параметр;
— коэффициент испарения феромона ρ — изменяющийся параметр;
— число дней/итераций $days$ — изменяющийся параметр;
— эталонный результат $ideal$;
— ошибка (разница) $mistake$ между действительным значением и эталонным.

4.4.1 Класс данных

Класс данных представляет собой набор из трёх матриц стоимостей в диапазоне от 1 до 10 для 10 вершин:

$$M_1 = \begin{pmatrix} 0 & 1 & 2 & 6 & 6 & 6 & 9 & 2 & 5 & 3 \\ 3 & 0 & 4 & 9 & 5 & 6 & 2 & 2 & 8 & 6 \\ 1 & 1 & 0 & 5 & 7 & 8 & 1 & 5 & 4 & 6 \\ 4 & 2 & 1 & 0 & 9 & 5 & 1 & 6 & 7 & 4 \\ 2 & 6 & 3 & 4 & 0 & 6 & 3 & 2 & 3 & 6 \\ 8 & 9 & 4 & 1 & 1 & 0 & 2 & 2 & 7 & 3 \\ 5 & 5 & 3 & 1 & 7 & 1 & 0 & 8 & 8 & 3 \\ 8 & 5 & 1 & 6 & 5 & 7 & 3 & 0 & 6 & 9 \\ 6 & 3 & 4 & 5 & 3 & 8 & 7 & 2 & 0 & 6 \\ 2 & 1 & 3 & 6 & 7 & 7 & 6 & 3 & 3 & 0 \end{pmatrix} \tag{1}$$

$$M_2 = \begin{pmatrix} 0 & 6 & 1 & 4 & 3 & 8 & 1 & 8 & 9 & 1 \\ 2 & 0 & 2 & 9 & 4 & 1 & 6 & 3 & 6 & 2 \\ 8 & 8 & 0 & 2 & 9 & 5 & 1 & 3 & 8 & 7 \\ 2 & 4 & 7 & 0 & 2 & 6 & 1 & 3 & 8 & 8 \\ 2 & 3 & 1 & 9 & 0 & 2 & 8 & 6 & 5 & 5 \\ 2 & 7 & 2 & 3 & 5 & 0 & 5 & 8 & 2 & 7 \\ 7 & 9 & 1 & 1 & 2 & 6 & 0 & 3 & 5 & 9 \\ 3 & 7 & 7 & 2 & 4 & 7 & 4 & 0 & 6 & 8 \\ 4 & 9 & 6 & 8 & 6 & 6 & 4 & 7 & 0 & 5 \\ 8 & 8 & 9 & 4 & 8 & 2 & 3 & 9 & 8 & 0 \end{pmatrix} \tag{2}$$

$$M_3 = \begin{pmatrix} 0 & 6 & 4 & 7 & 4 & 9 & 7 & 4 & 8 & 2 \\ 7 & 0 & 7 & 7 & 9 & 9 & 3 & 2 & 2 & 3 \\ 7 & 8 & 0 & 1 & 5 & 2 & 7 & 4 & 5 & 2 \\ 4 & 6 & 9 & 0 & 4 & 2 & 4 & 6 & 6 & 8 \\ 9 & 9 & 7 & 4 & 0 & 2 & 7 & 2 & 7 & 7 \\ 6 & 2 & 8 & 2 & 9 & 0 & 1 & 5 & 4 & 7 \\ 6 & 6 & 5 & 4 & 9 & 1 & 0 & 7 & 4 & 4 \\ 1 & 2 & 4 & 4 & 6 & 3 & 7 & 0 & 4 & 8 \\ 2 & 6 & 3 & 3 & 5 & 9 & 7 & 5 & 0 & 3 \\ 8 & 2 & 3 & 7 & 9 & 4 & 6 & 3 & 7 & 0 \end{pmatrix} \quad (3)$$

Для данного класса данных приведена таблица 1 с выборкой 10 параметров, которые наилучшим образом решают поставленную задачу. Ошибок в среднем — значение, указывающее на среднюю разницу идеального и найденного оптимального пути во время процесса параметризации класса данных.

Таблица 1 – Результат выборки параметров

α	ρ	Итерации, шт.	Среднее значение ошибки
0.4	0.2	300	0
0.4	0.7	300	0
0.7	0.7	300	0
0.7	0.7	500	0
0.1	0.3	300	0
0.1	0.4	400	0.333
0.1	0.9	500	0.333
0.2	0.1	200	0.333
0.2	0.3	500	0.333
0.3	0.1	500	0.333

ЗАКЛЮЧЕНИЕ

Цель работы достигнута: проведено исследование лучших и худших случаев работы алгоритмов поиска подстроки в строке стандартным алгоритмом и Кнута – Морриса – Пратта.

В ходе выполнения лабораторной работы были решены все задачи:

- описаны используемые алгоритмы поиска;
- выбраны средства программной реализации;
- реализованы данные алгоритмы поиска;
- проанализированы алгоритмы по количеству сравнений;
- подготовлен отчет по лабораторной работе.

В результате анализа замеров времени выполнения был сделан вывод, что, начиная с размера текста, равного 1000, алгоритм КМП работает быстрее стандартного алгоритма. Для текста размера меньшего, чем 256 символов, рекомендуется использовать стандартный алгоритм.